

Pilot -Autopilot Interaction: A Formal Perspective

Asaf Degani

NASA Ames Research Center
Moffett Field, CA 94035-1000, USA
adegani@mail.arc.gov

Michael Heymann

Department of Computer Science
Technion, Haifa 32000, Israel
heymanncs@cs.technion.ac.il

ABSTRACT

This paper discusses a formal perspective to the analysis of user interaction with machines, in general, and pilot interaction with automated flight control systems, in particular. It addresses the issue of correct interaction between the user and the machine by asking whether the information provided to the user about the machine, and the display of this information, enables the user to perform his or her tasks reliably and successfully. We explain this perspective by looking at one example of pilots' interaction with a modern autopilot.

Keywords

Automation, formal methods, design methods.

INTRODUCTION

In recent years, pilot interaction with automated flight control systems has become a major concern in the transport industry [6]. This problem has variously been termed as lack of mode-awareness, mode-confusion, or automation-surprises [17]. Two factors are repeatedly cited in accident and incident reports and in the scientific literature as being responsible for such breakdowns: (1) The interface between the user and the machine provides inadequate information about the status of the machine [2, 8, 12]; (2) The user has an inadequate "mental model" of the machine's behavior [10, 15]. Both factors may limit the user's ability to anticipate reliably the next configuration (e.g., mode) of the machine, and hence may lead to false expectations, confusion, and error [4]. In this paper, we shall focus on the first factor – the inadequacy of the interface.

In high-risk systems, such as commercial aviation, faulty interaction of the user with the machine has led to catastrophic results [11]. This faulty interaction has been variously attributed to a combination of human and machine problems. However, the distinction between human error, inadequate training, lack of situation awareness, and interface design errors is blurred (see e.g. [1]). One aspect is the complexity of automatic control systems and the lack of rigorous methods for their systematic analysis and evaluation [14].

The objective of the present paper is to suggest a formal perspective for examining human-automation interaction. The paper is organized as follows: We first discuss three elements that are part of this interaction – the interface, the user's model, and the task – and present several possible discrepancies among these elements. Following this discussion, we describe the vertical flight modes in the autopilot of a modern commercial aircraft, and present an incident involving these modes. We then analyze in detail, the pilots' interaction with the autopilot and identify discrepancies that lead to such mishaps.

Formal representation

A given machine, for example an autopilot, has a variety of modes. Each mode defines a specific behavior (e.g., "Vertical-Speed", "Altitude Hold", "Glide-Slope", etc. in a modern autopilot). The pilot must interact with this autopilot in order to perform a specified task (e.g., climb to 9,000 feet at a rate of 2,500 feet per minute, maintain 9,000 feet, or descend toward the runway). In the interaction between the user and the machine, three *elements* play a major role: (1) The set of tasks, or *task-specifications*, that the user must perform in order to operate the machine (e.g., engage "Vertical Speed" mode and note automatic transitions among modes); (2) The *user-model* of the machine's behavior, (e.g., what the pilot knows about how the aircraft behaves from training and from aircraft manuals) and (3) the *interface* through which user obtains information about the machine's behavior (e.g., the flight mode annunciator). These three elements must be suitably matched in order to insure correct and reliable user-machine interaction.

Figure 1 describes these elements. Each of the three circles represents the region where one of the elements is, by itself, “adequate”. For the purpose of this discussion, we shall always assume that the machine-behavior and the task-specification are given, valid, and correct. Therefore, we are interested in considering the interplay between the *user-model* and *interface* with respect to the machine-behavior and task-specification.

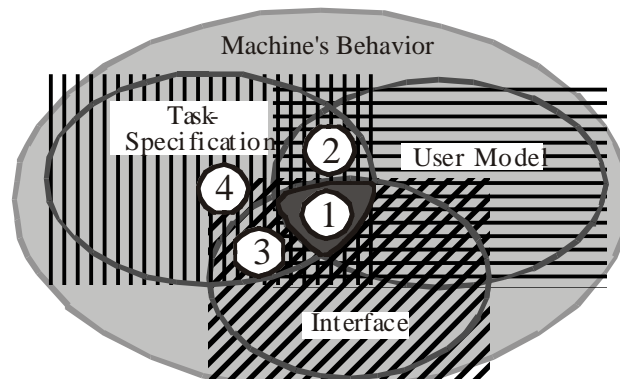


Figure 1. Interrelations.

Region 1 represents the situation in which all three elements are adequate, and correct interaction is possible. Region 2 represents the situation in which the task-specification and the user-model are adequate, but the interface is inadequate. Region 3 represents the situation in which the task specification and the interface are adequate, but the user-model is not. Finally, region 4 represents the case in which both the interface and the user-model are inadequate for the task.

Let's consider all the possible discrepancies between the machine's behavior, the user-model, and the interface in relation to a given task that the user must perform. To describe such discrepancies in a precise and formal (i.e., mathematical) way, we need some method, or language, for representation. More than 80 percent of the computer-code in modern Automatic Flight Control Systems (AFCS) consists of logical statements (the rest consists of continuous equations). A discrete representation of states and events is a natural medium for describing the behavior of autopilots and other components of automatic flight control systems (see e.g. [16]). This representation is easily extended to also represent the mode annunciation on the interface [9,13], the user's task (specifications) as well as the user's model of the machine's behavior.

A basic fragment of such a representation is a *state transition* that can be used to describe a statement such as, “If the user executes event α when the machine is in state 1 and condition C is TRUE, then the system transitions to state 2.” Many of the informal statements concerning the behavior of automated control systems are of this nature: “If the pilot pushes button x when the aircraft is in CRUISE mode and the descent profile is armed, then the aircraft transitions to the DESCENT mode.” A discrete event representation such as the Finite State Machine theory and its corresponding state-transition-diagrams are the formal mechanism for collecting such fragments into a whole [5].

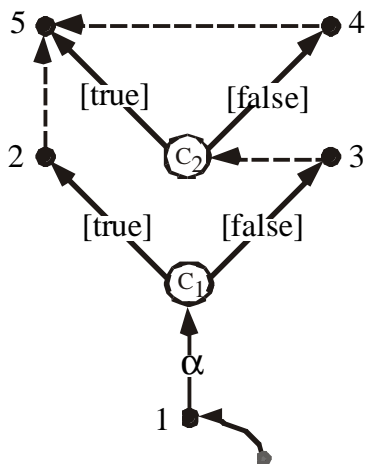
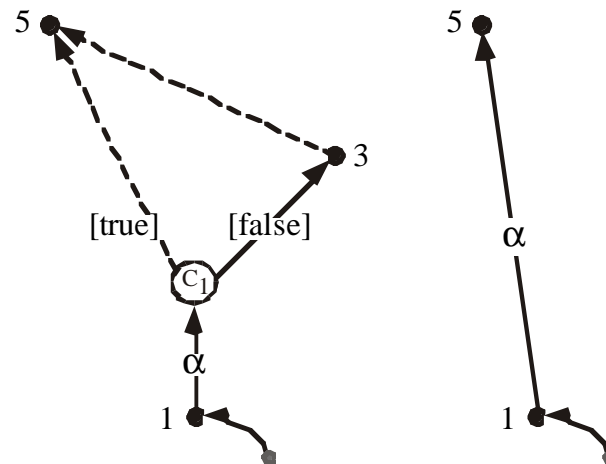


Figure 2. Simple machine (complete model)

Figure 2 is a State Transition Diagram of a simple machine. The machine starts at state 1, and upon execution (by the user) of event “ α ”, moves along to either state 2 or state 3, according to whether the condition “ C_1 ” is TRUE or FALSE. The dashed lines represent transitions that take place automatically and instantaneously. Thus, if state 2 is reached, the system moves to state 5 immediately, while if it reaches state 3, it moves to either state 5 or to state 4 depending on whether condition “ C_2 ” is TRUE or FALSE.

Suppose that the task specification is to drive the system from state 1 to state 5, via state 3 (because state 2 may cause some undesired behavior, e.g., deploying thrust-reversers in flight). To do this correctly, it is necessary to know whether condition “ C_1 ” is TRUE or FALSE, because only FALSE will lead us to state 5 via state 3. However, since we do not care whether state 4 is visited or not, condition “ C_2 ” is irrelevant for us. Figure 3(a) is a reduced, yet adequate, user-model for this machine.



Figure

3(a).

Figure 3(b)

Now let's consider the interaction between the *interface* and the *user-model* and relate this to the regions in Figure 1. The machine we are still dealing with is the one described in Figure 2, and our task specification is to drive the system from state 1 to state 5, via state 3. The first case is that our operator (e.g., pilot) has a correct user-model, but is *not* provided with an adequate interface (e.g., indicating whether C_1 is true or false). This situation, in which the *user-model* is adequate but the *interface* is inadequate, is represented by region 2 of Figure 1. Clearly, in this case, correct task execution cannot be guaranteed. In the second case, the pilot has an inadequate user-model. That is, the pilot is not told about the existence of condition C_1 , and therefore his or her user-model looks like Figure 3(b). According to this *user-model*, initiating event α *always* drives the machine to state 5. A correct display that indicates the status of condition C_1 would not be of any help here! The user would not be able to relate the status of condition C_1 (TRUE, FALSE) to the behavior of the machine. This situation, in which the *user-model* is inadequate but the *interface* is adequate, is represented by region 3 of Figure 1. Obviously, correct task execution cannot be guaranteed here. Finally, the worst case occurs when both the *user-model* and the *interface* are inadequate for the task (Region 4).

AUTOPILOT EXAMPLE

In this section we shall describe a specific autopilot and its behavior, show the interface, and provide an excerpt from an incident that took place while using this autopilot. Specifically, we shall address pilot interaction with this autopilot during the capture maneuver in which the autopilot transitions from climb or descent to level flight (see [3] for a formal and model-based analysis of this example).

Behavior

The sequence of actions in using this autopilot to climb to a higher altitude goes like this: The aircraft is at some initial altitude (say 5,000 feet). The pilot selects an altitude to which the autopilot should climb and level-off at (e.g., 9,000 feet). The pilot then engages a climb mode such as “vertical speed”, and the aircraft starts to climb. (Figure 4 is a profile describing this scenario.) The current altitude of the aircraft is depicted in the pointed box in the lower left corner and the selected altitude is depicted in the upper-right box.

Somewhere prior to reaching 9,000 feet, the autopilot automatically transitions to the “Capture” mode and starts the (altitude) capture maneuver. In Figure 4 the capture-start altitude is 7,000 feet. Above 7,000 feet the autopilot commands a parabolic trajectory in which the climb-rate varies as the aircraft gradually transitions to level flight (at 9,000). This trajectory depends on the aircraft's climb-rate, altitude, and acceleration. The main point is that the altitude, at which the transition to the “Capture” mode takes place, varies. The pilot has no pre-knowledge of this altitude.

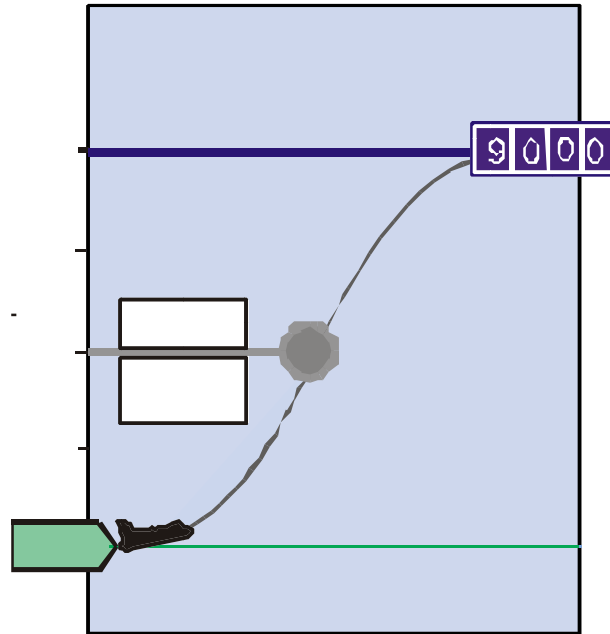


Figure 4. Climb profile

Interface

Figures 5(a) and 5(b) are schematic illustrations of the “Guidance Control Panel” (GCP) and “Electronic Attitude Display Indicator (EADI)” of this autopilot, respectively.

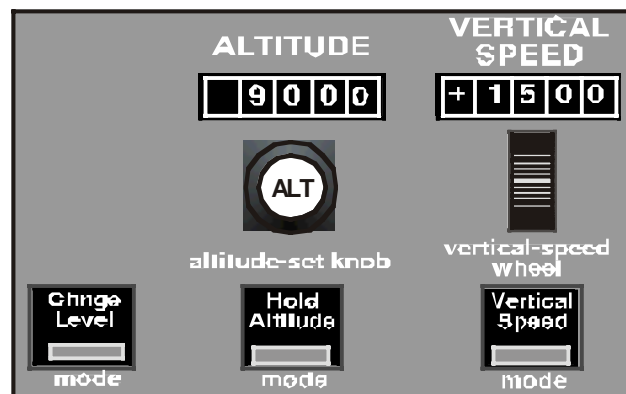


Figure 5(a). Guidance Control Panel.

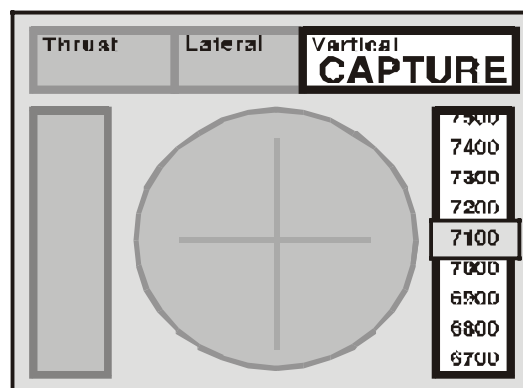


Figure 5(b). Electronic Attitude Display Indicator.

The GCP illustrated in Figure 5(a) includes buttons for engagement (by the pilot) of climb/descend and hold-altitude modes. In the top portion of the GCP is a window indicating the selected altitude setting. The pilot can reset the altitude setting by rotating the altitude knob.

Information on the current mode of the aircraft is provided on the “Flight Mode Annunciator” located on the top portion of the EADI. In Figure 5(b) the current vertical mode, displayed in the right-most window, is “Capture” (the “Thrust”

the current altitude of the aircraft, is displayed on the right side of the EADI. By viewing both the GCP and the EADI, the pilot has knowledge of the GCP altitude setting, the active vertical mode, and the current aircraft altitude at any time.

Incident

The following is an altitude deviation incident, involving this autopilot, which was reported to NASA's Aviation Safety Reporting System (ASRS). An altitude deviation (or "altitude bust" in aviation jargon) is a situation in which the aircraft, for whatever reason, does not level off at the intended altitude:

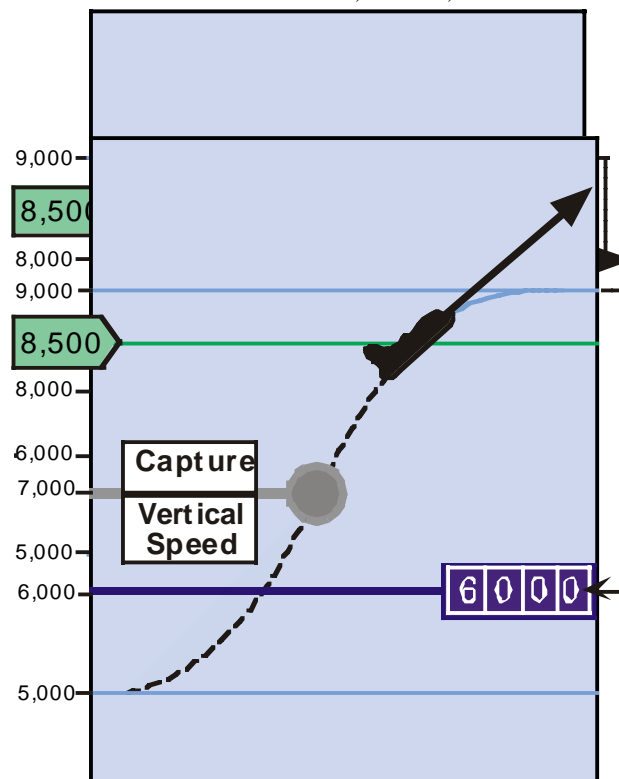
On climb to 27,000 feet and leaving 26,500 feet. Memphis Center gave us a clearance to descend to 24,000 feet. The aircraft had gone to "Capture" mode when the first officer selected 24,000 feet on the GCP altitude setting... and the aircraft continued to climb at approximately 300 feet-per-minute. There was no altitude warning and this "altitude bust" went unnoticed by myself and the first officer, due to the slight rate-of-climb. At 28,500, Memphis Center asked our altitude and I replied 28,500 and started an immediate descent to 24,000 feet (ASRS report # 113722).

ANALYSIS

We shall now examine pilot interaction with this autopilot while in "Capture" mode. The action that we will consider here is setting the altitude window to a value that is behind the aircraft – just as it happened in the incident. We are climbing toward 27,000, the aircraft is now at 26,500 feet, and at this moment we get a clearance to descend back to 24,000 feet. 24,000 is lower than 26,500 and is *behind* the direction and current altitude of the aircraft. Let us consider a similar situation at somewhat different altitudes. We are at 5,000 feet climbing to 9,000 in "Vertical Speed" mode. At 7,000 the autopilot transitions from "Vertical Speed" mode (which was initially engaged by the pilot) to "Capture" mode (which was automatically engaged by the autopilot). The aircraft starts the parabolic trajectory to capture 9,000 feet. Now at an altitude of 8,500 feet we get a *new* clearance. Let us see what the aircraft will do: If we get a clearance to descend and maintain 6,000 feet and we set this value in the GCU altitude window, the aircraft will *not* honor this setting and, instead, continue its climb at the current vertical speed, indefinitely. This behavior is called "kill the capture" -- and kill it, it does. This is exactly what happened in the above-mentioned incident. The profile depicted in Figure 6 describes this behavior.

Figure 6. Set altitude to "below" capture-start.

On the other hand, if we get a clearance to descend and maintain 8,000 feet, and we set this value in the GCU altitude



window, the aircraft *will* honor this setting, dive down and capture 8,000 feet (Figure 7).

Figure 7. Set altitude to "above" capture-start.

current climb-rate and “kill the capture”. In the second case the aircraft will capture the newly set altitude. So how can the pilot anticipate what the aircraft will do in each case?

It turns out that there is a hidden condition here, that revolves around the altitude at which the aircraft transitions to the “Capture” mode (7,000 feet in our example): If the newly set altitude is *below* (e.g., 6,000 feet) the “start capture altitude”, then the aircraft *will* kill the capture. But if the newly set altitude is *above* (e.g., 8,000 feet) the “start capture altitude”, the aircraft *will* capture the specified altitude.

Do pilots know about this behavior and the condition? No. It’s not in the manual nor is it mentioned in ground school or Initial Operating Experience (IOE) -- it is practically unknown! Now we are not talking about some minor deficiency, but a critical maneuver that sometimes takes place very close to the ground. The pilots’ user-model is inadequate for the task of capturing an altitude after resetting the altitude to a value behind the aircraft. If we go back to our earlier discussion, we have a condition in which the user-model is inadequate for the task. This deficiency corresponds to region 3 in Figure 1.

Now, let’s say that we have provided this information to the flight crews and thereby updated their user-model to include the condition (“altitude capture start”) and the two possible outcomes of setting the new altitude. Presumably now we should be out of Region 3 and in Region 1 and everything should be OK.

Yes?

Actually, not. The display is also inadequate for the task – and here is why: To resolve what the aircraft will do we need to know the altitude at which the autopilot transitions to capture (e.g., 7,000 feet). But in practice, it is almost impossible to obtain this value with the current display. First, the pilot has no preview of this value and the interface does not display it. Secondly, the transition to the “Capture” mode happens automatically. In order to obtain the altitude at which the autopilot transitions to “Capture”, the pilot must “hunt” for the automatic transition, and at that very moment look down to the altitude tape on the interface and catch the aircraft altitude as it rolls by. Thirdly, this altitude value is not retained by the display; once the transition takes place, the value is gone and there is no way to retrieve it. For all practical purpose, it is impossible to reliably obtain this value. The current display is indeed inadequate for the task.

To recap, the situation just described actually corresponds to Region 4 in Figure 1 where both the *user-model* and the *interface* are inadequate for the task: the pilots do not know about the start-capture condition and the interface does not provide the start-capture value. Attempts to solve the problem by updating the *user-model* to include this condition will get us out of Region 4 into – Region 2. Yes, the pilot is now aware of the condition, but the interface does not provide the capture-start value; the task-specification and the user-model are now adequate, but the interface is, still, *inadequate*. To this end, the interface is incorrect and correct task execution cannot be guaranteed.

And here we are stuck. The only way to remove this deficiency is by modifying the interface: either by providing the value itself (which will still require the pilot to perform a mental calculation to resolve what the aircraft will do) or some qualitative indicator that will resolve whether or not capture will take place. Finally, one can also redesign the autopilot behavior, but that, of course, is beyond the scope of this paper.

CONCLUSIONS

There are two kinds of reactions to such problems –that is, after the pilots become confused. Some pilots blame it on themselves, on being inattentive and out-of-the-loop. Others blame it on the autopilot, write it up in the log as a malfunction and demand that a mechanic fix the autopilot.

But the problem, of course, is neither of the above: it’s not about being inattentive and the autopilot always checks OK. Such interface inadequacy limits pilots’ ability to interact reliably with the autopilot and generates confusion, apprehension, and mistrust in automation. Contrary to common belief, these types of problems, which unfortunately are not rare in present-day automated control systems, cannot be sufficiently resolved with further training, improved situation awareness, or by updating aircraft manuals.

The deeper issue is not about this specific autopilot or its interface – it’s about design and evaluation of automated control systems. The fact of the matter is that the people who designed it did their best to provide a sound interface, and the certification officials who approved it also evaluated it to the best of their abilities. Present-day automated control systems are very complex and user-interaction is, inevitably, complex as well; there is no escape from this reality, at least by today’s standards.

The naked truth is that current methods for designing and evaluating human-automation interactions are, in themselves, inadequate. Current methods, which consist of “what-if” questions, cognitive walk-through, simulations, and experimentation, are simply not systematic and rigorous enough to cover all possible pilot-automation interactions. This limitation is what lead to the design deficiency that we described in this paper. There is an urgent need for better design and evaluation methods from avionics vendors, airframe manufacturers, and certification officials [14].

The formal approach discussed in this paper is a way to start considering such human-automation problems from a more systematic perspective. A perspective that takes into account the actual behavior of the automation and the specified

interface, and task specification elements. Such a rigorous and systematic methods for designing and evaluating large and complex human-automation interaction are indeed feasible. Initial steps towards the development of such design methods have recently been proposed [7]. The increasing complexity of the automated systems, the lack of systematic design/evaluation methods, and the diminishing tolerance to catastrophic failures and its intolerably high cost urge us to be satisfied with nothing less.

ACKNOWLEDGEMENTS

This work was conducted as part of NASA's base research and technology effort, human-automation theory sub-element. The authors were supported by Grant NCC 2-798 from the NASA Ames Research Center to the San Jose State University. We gratefully acknowledge the support of one aerospace firm and its staff for providing extensive simulator-time and resources necessary to build and validate the models that underlie this analysis. Special thanks are due to Captain David Austin for flying the simulator and providing helpful insights. The authors thank Rowena Morrison, Lance Sherry, Alex Kirlik, Immanuel Barshi, and Barry Crane for helpful comments on an earlier draft.

REFERENCES

1. Aviation Week and Space Technology. (1995). Automated cockpits: who s in charge? Part 1 and 2. 142(5 and 6),
2. Billings, C. E. (1997). Aviation automation: The search for a human centered approach. Hillsdale, NJ: Erlbaum.
3. Degani, A. and Heymann, M. (submitted). Some formal aspects of human-automation interaction.
4. Degani, A., Shafto, M., and Kirlik, A. (1999). Modes in human-machine systems: Review, classification, and application. *International Journal of Aviation Psychology*, 9(2), 125-138.
5. Harel, D. (1988). On visual formalisms. *Communications of the ACM*, 31(5), 514-530.
6. Funk, K., Lyall, B., Wilson, J., Vint, R., Niemczyk, M., Surotrguh, C., and Owen, G. (1999). Flight deck automation issues. *International Journal of Aviation Psychology*, 9(2), 109-124.
7. Heymann M., and Degani A. (forthcoming). Display synthesis and interface resolution for interactive discrete event systems.
8. Indian Court of Inquiry. (1992). Report on accident to Indian Airlines Airbus A-320 aircraft VT-EPN at Bangalore on 14th February 1990. Indian Government.
9. Jacob, R. J. K. (1986). A specification language for direct-manipulation user interface. *ACM Transactions on Graphics*, 5(4), 283-317.
10. Javaux, D., and De Keyser, V. (1998). The Cognitive Complexity of Pilot-Mode Interaction: A Possible Explanation of Sarter and Woods' Classical Result. In G. Boy, C. Graeber and J. Robert (Ed.), *Proceeding of the International Conference on Human-Computer Interaction in Aeronautics Conference* (pp. 49-54). Montreal, Quebec: Ecole Polytechnique de Montreal.
11. Mellor, P. (1994). CAD: Computer aided disasters. *High Integrity Systems*, 1(2), 101-156.
12. Norman, D. A. (1990). The 'problem' with automation: inappropriate feedback and interaction, not 'over-automation.' *Phil. Trans. Research Society London*, B 327, 585-593.
13. Parnas, D. (1969). On the use of transition diagrams in the design of a user interface for an interactive computer system. *Proceeding of the 24th Annual ACM Conference* (pp. 379-385).
14. Radio Technical Commission for Aeronautics (RTCA). (1999). Report of Task Force 4 — Certification. Executive summary and portions of the report of the report are available at the following URL -- <http://www.rtca.org/>. Washington, DC: RTCA.
15. Sarter, N. B., and Woods, D. D. (1995). How in the world did I ever get into that mode? Mode error and awareness in supervisory control. *Human Factors*, 37(1), 5-19.
16. Sherry, L., and Polson, P. (1999). Shared models of flight management system vertical guidance. *International Journal of Aviation Psychology*, 9(2), 139-154.
17. Woods, D., Sarter, N., and Billings, C. (1997). Automation surprises. In G. Salvendy (Ed.), *Handbook of human factors and ergonomics* (pp. 1926-1943). New York: John Wiley.